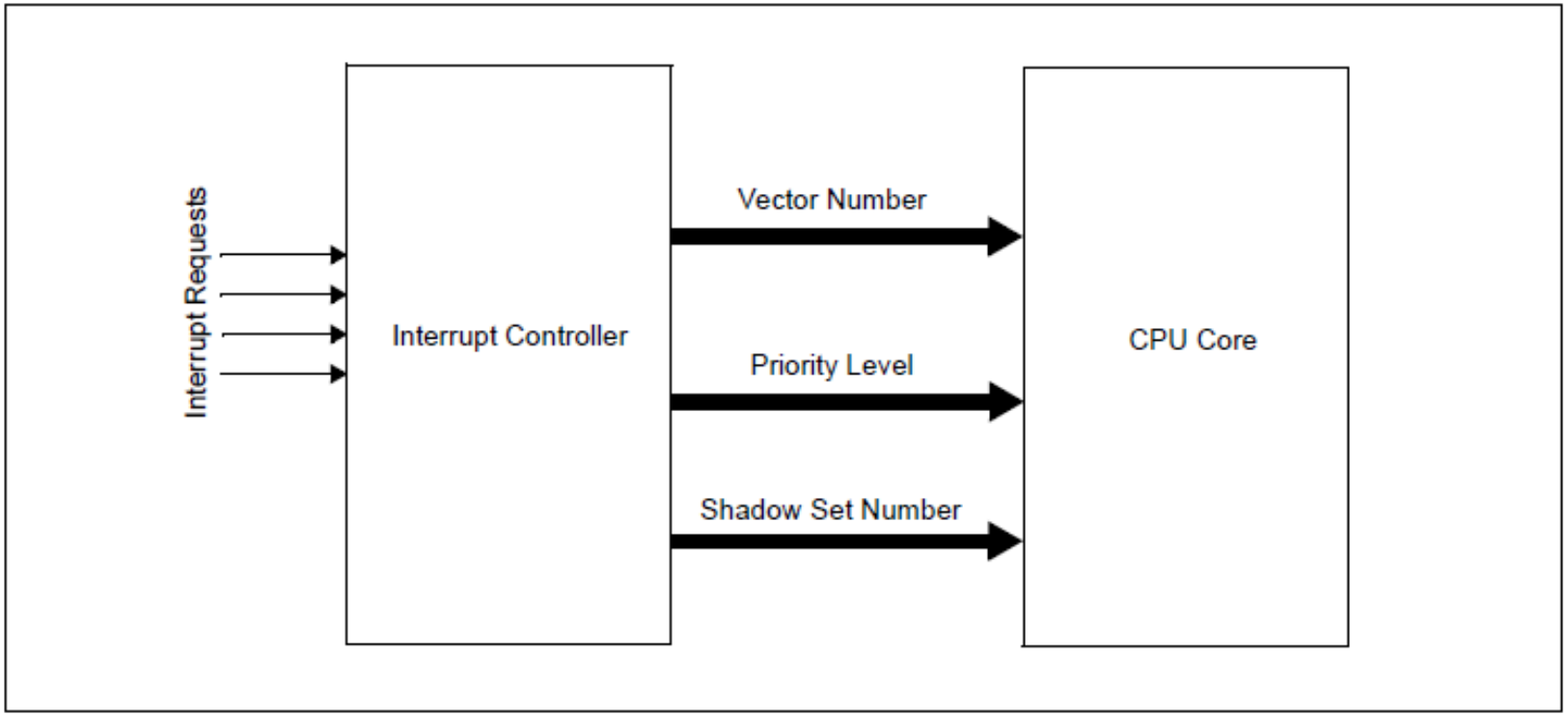
# PIC Interrupt Service Routines

**Dr. Edward Nava**
**ejnava@unm.edu**

# PIC Interrupt Controller

**FIGURE 7-1: INTERRUPT CONTROLLER MODULE**



# Interrupt Service Routines

- You must have a specific function that is invoked and executed when the interrupt occurs. (Interrupt Service Routine – ISR)
- In the multi-vector mode, you must have separate ISRs for each of the interrupt sources you wish to use.

# Interrupt Vectors

**TABLE 7-1: INTERRUPT IRQ AND VECTOR LOCATION**

| Interrupt Source(1) | IRQ | Vector Number | Interrupt Bit Location | | | |
|---|---|---|---|---|---|---|
| Highest Natural Order Priority | | | Flag | Enable | Priority | Subpriority |
| CT – Core Timer Interrupt | 0 | 0 | IFS0<0> | IEC0<0> | IPC0<4:2> | IPC0<1:0> |
| CS0 – Core Software Interrupt 0 | 1 | 1 | IFS0<1> | IEC0<1> | IPC0<12:10> | IPC0<9:8> |
| CS1 – Core Software Interrupt 1 | 2 | 2 | IFS0<2> | IEC0<2> | IPC0<20:18> | IPC0<17:16> |
| INT0 – External Interrupt 0 | 3 | 3 | IFS0<3> | IEC0<3> | IPC0<28:26> | IPC0<25:24> |
| T1 – Timer1 | 4 | 4 | IFS0<4> | IEC0<4> | IPC1<4:2> | IPC1<1:0> |
| IC1 – Input Capture 1 | 5 | 5 | IFS0<5> | IEC0<5> | IPC1<12:10> | IPC1<9:8> |
| OC1 – Output Compare 1 | 6 | 6 | IFS0<6> | IEC0<6> | IPC1<20:18> | IPC1<17:16> |
| INT1 – External Interrupt 1 | 7 | 7 | IFS0<7> | IEC0<7> | IPC1<28:26> | IPC1<25:24> |
| T2 – Timer2 | 8 | 8 | IFS0<8> | IEC0<8> | IPC2<4:2> | IPC2<1:0> |
| IC2 – Input Capture 2 | 9 | 9 | IFS0<9> | IEC0<9> | IPC2<12:10> | IPC2<9:8> |
| OC2 – Output Compare 2 | 10 | 10 | IFS0<10> | IEC0<10> | IPC2<20:18> | IPC2<17:16> |
| INT2 – External Interrupt 2 | 11 | 11 | IFS0<11> | IEC0<11> | IPC2<28:26> | IPC2<25:24> |
| T3 – Timer3 | 12 | 12 | IFS0<12> | IEC0<12> | IPC3<4:2> | IPC3<1:0> |
| IC3 – Input Capture 3 | 13 | 13 | IFS0<13> | IEC0<13> | IPC3<12:10> | IPC3<9:8> |
| OC3 – Output Compare 3 | 14 | 14 | IFS0<14> | IEC0<14> | IPC3<20:18> | IPC3<17:16> |

Ref: PIC Data Sheet

# ISR Usage Requirements

Interrupts can occur at any time, making the ISR execution asynchronous with respect to the main program. Because of this, there are several constraints.

1. An ISR cannot be called directly by any other function.
2. No arguments can be passed to an ISR.
3. An ISR cannot return any values.
4. Ideally, an ISR should not call any other functions.

# Writing an ISR

- A function is marked as a handler function via either the interrupt attribute or the interrupt pragma1.
- Each method is functionally equivalent to the other.
- The interrupt is specified as handling interrupts of a specific priority level or for operating in single vector mode.

INTERRUPT ATTRIBUTE

__attribute__((interrupt([IPL*n*[SRS|SOFT|AUTO]])))

Where *n* is in the range of 0..7, inclusive.

INTERRUPT PRAGMA

# pragma interrupt *function-name* single [vector [@] 0

Where *n* is in the range of 0..7, inclusive. The IPL*n* specifier may be all uppercase or all lowercase.

# Function Definition for Handler Function

- The function definition for a handler function indicated by an interrupt pragma must follow in the same translation unit as the pragma itself.
- The interrupt attribute will also indicate that a function definition is an interrupt handler. It is functionally equivalent to the interrupt pragma.
- For example, the definitions of foo below both indicate that it is an interrupt handler function for an interrupt of priority 4 that uses software context saving.

```
#pragma interrupt foo IPL4SOFT
void foo (void)
```

is functionally equivalent to

```
void __attribute__ ((interrupt(IPL4SOFT))) foo (void)
```

# ASSOCIATING A HANDLER FUNCTION WITH AN EXCEPTION VECTOR

- An interrupt handler function can be associated with an interrupt vector

  The following declaration specifies that function foo will be created as an interrupt handler function of priority four. foo will be located at the address of exception vector 54.

  ```
  void __attribute__ ((interrupt(ipl4))) __attribute__ ((at_vector(54)))
  foo (void)
  ```

  The following pragma specifies that function bar will be created as an interrupt handler function of priority five. bar will be located in general purpose program memory (.text section). A dispatch function targeting bar will be created at exception vector address 23.

  ```
  #pragma interrupt bar ipl5 vector 23
  ```

# __ISR Macros

- The <sys/attribs.h> header file provides macros intended to simplify the application of attributes to functions. There are also vector macros defined in the processor header files.

__ISR(v, ipl)

Use the __ISR(v, ipl) macro to assign the vector-number location and associate it with the software-assigned interrupt priority. This will place a jump to the interrupt handler at the associated vector location.

void __ISR (_TIMER_1_VECTOR, IPL4) Timer1Handler (void);

This creates an interrupt handler function for the Timer 1 interrupt that has an interrupt priority level of four. The compiler places a dispatch function at the vector location associated with macro _TIMER_1_VECTOR as defined in the device-specific header file. To reach this function, the Timer 1 interrupt flag and enable bits must be set, and the interrupt priority should be set to a level of four.

# Sample ISR

```
void __ISR(_TIMER_2_VECTOR, ipl2) Timer2Handler(void)
{
        // clear the interrupt flag
        mT2ClearIntFlag();
          PORTToggleBits(IOPORT_B,BIT_10);
}
```

# Initialization Code in main()

```
void DelayInit()
{
        unsigned int tcfg;

        /* Config Timer 2. This sets it to count 39,062 Hz with a period of T2_TICK */

        tcfg = T2_ON|T2_SOURCE_INT|T2_PS_1_256;
        OpenTimer2(tcfg, T2_TICK);

        /* Now enable system-wide multi-vector interrupt handling     */
        INTEnableSystemMultiVectoredInt();

        /* configure timer 2 interrupt with priority of 2          */
        ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_2);

        /* Clear interrupt flag                */
        mT2ClearIntFlag();
}
```